Vietnam J Comput Sci (2015) 2:121–131
DOI 10.1007/s40595-014-0037-2

REGULAR PAPER

Reduction of function evaluation in differential evolution using nearest neighbor comparison

Hoang Anh Pham

Received: 5 June 2014 / Accepted: 5 December 2014 / Published online: 18 December 2014

Abstract Approximation models have recently been introduced to differential evolution (DE) to reduce expensive fitness evaluation in function optimization. These models basically require additional control parameters and/or external storage for the learning process. Depending on the choice of the additional parameters, the strategies may have different levels of efficiency. The present paper introduces an alternative way for reducing function evaluations in differential evolution, which does not require additional control parameter and external archive. The algorithm uses a nearest neighbor in the search population to judge whether a new point is worth evaluating, so that unnecessary evaluations can be avoided. The performance of this new scheme of differential evolution, known as differential evolution with nearest neighbor comparison (DE-NNC), is demonstrated and compared with that of standard DE as well as approximation models including differential evolution using k-nearest neighbor predictor (DE-kNN), differential evolution using speeded-up k-nearest neighbor estimator (DE-EkNN) and DE with estimated comparison method through some test functions. The results show that DE-NNC can produce considerable reduction of actual function calls compared to DE and is competitive to DE-kNN, DE-EkNN and DE with estimated comparison.

Keywords Global optimization · Differential evolution · Nearest neighbor · Function evaluation reduction

H. A. Pham (✉)
Department of Structural Mechanics, National University of Civil Engineering, 55 Giai Phong Road, Hanoi, Vietnam
e-mail: anhpham.nuce@gmail.com

1 Introduction

Differential evolution (DE), which was introduced by Storn and Price [1] is a population-based optimizer. DE creates a trial individual using differences within the search population. The population is then restructured by survival individuals evolutionally. The algorithm is simple, easy to use and has shown better global convergence and robustness than most other genetic algorithms, suitable for various optimization problems [2]. However, like other population-based algorithms such as genetic algorithms (GA) and particle swarm optimization (PSO), one of the main issues in applying DE is its expensive computation requirement. This is due to the fact that evolutionary algorithm (EA) often needs to evaluate objective function thousand times to get optimal solutions. It becomes more pronounced when the cost of function evaluation becomes higher.

Research in reducing the computational burden in EA has been focusing on using function approximations, so-called meta-model or surrogate model [3–6]. Some of the popular approximation models in evolutionary computation are quadratic models [7], kriging models [7,8], neural network models [9] and radial basis function (RBF) network models [10–13]. In these approximation strategies, objective function is estimated by approximation model and the optimization problem is solved utilizing the approximated values. The effectiveness of this strategy depends largely on the accuracy of the approximation model. A time-consuming learning process is often invoked to obtain a high accuracy model. Thus, time-efficient approximate model is particularly important for expensive function evaluation.

Methods using fitness approximation based on meta-model have been recently introduced to differential evolution, including DE-kNN [14] and DE-EkNN [15]. Both methods use k-nearest neighbor (kNN) predictor constructed through

a dynamic learning to reduce the exact evaluation calls during DE search. In the early predefined iterations, the algorithm processes as usual, i.e., as standard DE procedure with exact objective function evaluations. All fitness values calculated are stored in an archive to be reused as training population. In later iterations, prediction function value computed from k-nearest samples in the training population is assigned to each solution. This new population is sorted (in the order of optimization) based on prediction function values. A chosen number of best solution values in the new population will be replaced by the exact function values and then stored in the training population. In DE-kNN, all re-evaluated samples are stored and the training population gradually increases. Two major differences between DE-EkNN and DE-kNN are that a weighted average kNN is used in DE-EkNN to estimate the fitness value for effective prediction and the archive is updated by selectively storing training samples for efficiency [15]. Thus, DE-EkNN has more compact archive than DE-kNN. Both DE-kNN and DE-EkNN have been shown through benchmark test functions to be able to converge towards the global optima with less actual evaluations. As pointed out by Park and Lee in their paper [15], two of the major drawbacks of the kNN predictor are the requirement of time to find the nearest neighbors and the need of memory storage to keep samples in the archive. This tends to become more pronounced as the dimension of the problem and the size of the archive increase. So the algorithms are not applicable to the problem whose dimension is extremely large, the real-time application which needs rapidness or an embedded device which lacks memory storage [15].

A new strategy for reducing the number of function evaluations is the estimated comparison method introduced by Takahama and Sakai [16,17]. In their method, they utilize a rough approximation model, which is an approximation model with low accuracy and without learning process to approximate the function values. The method is different from the surrogate models in that the rough approximated values are only used to estimate the order relation of two points. Function evaluations will be omitted if a point is judged as worse than the target point. The method of estimated comparison was first proposed with a potential model for function approximation [16] and shown to be efficient with much less evaluations compared to DE. The method also works well with other rough approximation models, including kernel average smoother and nearest neighbor smoother [17]. The efficiency of the estimated comparison is influenced by a parameter for error margin: lower value of error margin parameter can reject more trial individuals and omit a larger number of function evaluations, but can also increase the possibility of rejecting a good child; larger value reduces the possibility of rejecting a good child. However, the estimated comparison can reject fewer children and omit a small number of function evaluations. An improved estimated comparison is given by the same authors in [18], in which adaptive control was proposed to produce proper parameters to give more efficiency and stability. The estimated comparison was shown to be also effective for constraint optimization problem [19]. The advantage of DE with rough approximation-based comparison is that rough approximation is not too expensive and does not require the learning process and archive. The rough approximation is constructed on the current search population and only used for judgment, and the optimal solution is searched using the exact function values.

Introducing additional control parameters is required in both DE using kNN predictor and DE with estimated comparison method. DE-kNN and DE-EkNN need five and seven more parameters, respectively, whilst DE with estimated comparison adds one or two parameters. Proper parameters are often sought to ensure efficiency and stability. Good values of parameters are obtained by hand-tuning [14,17] or in an automatically adaptive way [15,18]. Obviously, more parameters bring more complexity to the algorithms.

The present paper proposes an alternative way to reduce function evaluation without additional control parameter. The proposed method applies comparison to judge whether or not a new child is worth evaluating, so that the evaluation of the objective function can be skipped. However, the method is different from the estimated comparison method. It uses the readily exact function value of a nearest neighbor in the population of the new child to compare with that of the parent, thus the approximation process is not necessary. The method is named as DE with the nearest neighbor comparison (DE-NNC) and can be viewed as another way of rough approximation-based comparison. The performance of DE-NNC is demonstrated through optimization of some test functions. The simulation results suggest that the proposed scheme is able to achieve good solutions and provide competitive reduction of the function evaluations compared to DE-kNN, DE-EkNN and DE with the estimated comparison method.

The organization of the rest of this paper is as follows. In Sect. 2, a brief introduction of DE is given. In Sect. 3, the main idea of the proposed DE-NNC is described in detail with a concept of *possibly useless trial* (PUT) and a *nearest neighbor comparison* method. Experiment results on some test functions are presented in Sect. 4. Finally, the conclusion is given in Sect. 5.

2 Basic of differential evolution

We search for the global optima of the objective function $f(\mathbf{x})$ over a continuous space $\mathbf{x} = \{x_i\}, x_i \in \left[x_{i,\min}, x_{i,\max}\right]$, $i = 1, 2, \ldots, n$. Classical differential evolution (DE) algorithm invented by Storn and Price [1] for this optimization problem is described in the following.

For each generation G, a population of NP parameter vectors $\mathbf{x}_k(G), k = 1, 2, \ldots, NP$, is utilized. The initial population is generated as

$$x_{k,i}(0) = x_{i,\min} + rand[0, 1]. (x_{i,\max} - x_{i,\min}), \quad i = 1, 2, \ldots, n \tag{1}$$

where *rand*[0,1] is a uniformly distributed random real value in the range [0,1]. For each target vector in the population $\mathbf{x}_k(G)$, $k = 1, 2, \ldots, NP$, a perturbed vector $\mathbf{y}$ is generated according to

$$\mathbf{y} = \mathbf{x}_{r_1}(G) + F.\left[\mathbf{x}_{r_2}(G) - \mathbf{x}_{r_3}(G)\right], \tag{2}$$

with r_1, r_2 and r_3 being randomly chosen integers and $1 \le r_1 \ne r_2 \ne r_3 \ne k \le NP$; F a real and constant factor usually chosen in the interval [0, 1], which controls the amplification of the differential variation $(\mathbf{x}_{r_2}(G) - \mathbf{x}_{r_3}(G))$.

Crossover is introduced to increase the diversity of the parameter vectors, creating a trial vector $\mathbf{z}$ with its elements determined by:

$$z_i = \begin{cases} y_i & \text{if } (rand[0, 1] \le Cr) \text{ or } (r = i) \\ x_{k,i(t)} & \text{if } (rand[0, 1] > Cr) \text{ and } (r \ne i) \end{cases} \tag{3}$$

Here, r is a randomly chosen integer from the interval [1, n]; Cr is a user-defined crossover constant from [0, 1]. The new vector $\mathbf{z}$ is then compared to $\mathbf{x}_k(G)$. If $\mathbf{z}$ yields a better objective function value, then $\mathbf{z}$ becomes a member of the next generation $(G + 1)$; otherwise, the old value $\mathbf{x}_k(G)$ is retained.

Basically, DE calls for objective function evaluation for every trial vector. It is desirable that trial vectors which might produce no better fitness should not be evaluated. In the following parts, we introduce the concept of *possibly useless trial* and employ a *nearest neighbor comparison* method to reduce useless computation.

3 Differential evolution with nearest neighbor comparison (DE-NNC)

The nearest neighbor comparison has the same strategy as the estimated comparison method by Takahama and Sakai [16], [17]. In the estimated comparison, a rough approximation model is used to estimate the order relation of two points. A child point $\mathbf{z}$ is judged better than the parent point $\mathbf{x}_k$ if the following condition is guaranteed:

$$\hat{f}(\mathbf{z}) < \hat{f}(\mathbf{x}_k) + \delta\sigma, \tag{4}$$

where $\hat{f}$ is the estimated function of f, σ is the error estimation of the approximation model and δ is a margin parameter for the approximation error. The parameter $\delta \ge 0$ controls the margin value for the approximation error. A lower value of δ can reject more trial individuals and omit a larger number of function evaluations, but can also increase the possibility of rejecting a good child; a larger value of δ reduces the possibility of rejecting good child. However, the estimated comparison can reject fewer children and omit a small number of function evaluations. Thus, the efficiency of the estimated comparison largely depends on the choice of the margin parameter. Different approximation models can be applied for estimated comparison, including the potential model and kernel smoother [17].

The nearest neighbor comparison, on the other hand, does not use function approximation and no additional control parameter is introduced. The details of the method are described in the following.

3.1 Concept of possibly useless trial

- A trial parameter vector with high possibility of having fitness worse than that of the current target vector is called a *possibly useless trial* vector (PUT vector).
- To judge whether a trial vector is a PUT vector, its nearest neighbor vector in the population is utilized to compare with the target vector. This method is named as the *nearest neighbor comparison* (NNC).

3.2 Nearest neighbor comparison method (NNC)

The step of NNC is give below:

- In the current population, a vector $\mathbf{x}_c(G)$ nearest to the considered trial vector is searched using the distance measure. For this task, Euclidean distance measured as in Eq. (5) is adopted. Other form of distance measurements such as Minkowsky metric can be used:

$$\mathrm{d}(\mathbf{x}, \mathbf{y}) = \sqrt{\left(\sum_{i=1}^{n} (x_i - y_i)^2\right)}, \tag{5}$$

  where $\mathrm{d}(\mathbf{x}, \mathbf{y})$ is the distance between two n-dimension vectors $\mathbf{x}$ and $\mathbf{y}$.
- The ready fitness of $\mathbf{x}_c(G)$ is then compared with that of the target vector $\mathbf{x}_k(G)$. If $f(\mathbf{x}_c(G))$ is worse than $f(\mathbf{x}_k(G))$, the trial vector possibly yields no better fitness than $\mathbf{x}_k(G)$, and it is judged as a PUT vector.
- Function evaluation will not be carried out for for the PUT vector.

The judgment by NNC is based on the fact that in the vicinity of a point in the search space, the objective function is often observed to behave monotonically, except points which are close to the local (global) optima. This judgment is illustrated in Fig. 1 for a simple minimization problem of

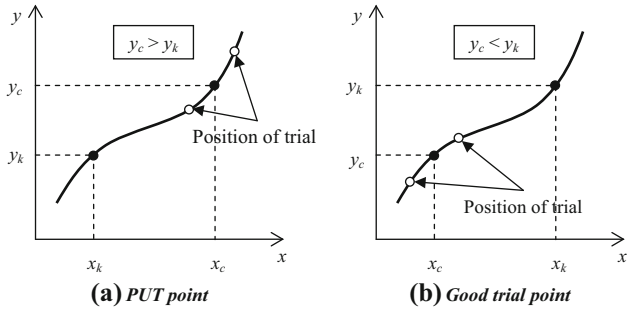


Fig. 1 Judgment of trial vector for minimization of single variable function

a single variable function $y(x) = f(x)$. Assume an increasing monotonic behavior of the function in the vicinity of the parent point, we observe that:

- Case 1 (Fig. 1a): trial point is on the right of x_k, x_c is worse than x_k, i.e., $y_c > y_k$. The trial point will be also worse thanx_k. So it is PUT point.
- Case 2 (Fig. 1b): trial point is on the left of x_k, x_c is better than x_k, i.e., $y_c < y_k$. The trial point will be better than x_k. So it is a good trial point.

Thus, in these cases, the trial point is completely judged by its nearest neighbor. For maximization problems, the comparison is reversed. The extension of the judgment to multi-dimension problems can be guaranteed as long as the monotonic property of the objective function is reserved. Therefore, wrong judgment might be made when a trial point is far away from the target point or the two points are in the vicinity of the local/global optima.

3.3 Algorithm description

The procedure of the proposed algorithm is basically similar to the conventional DE algorithm [1]. The nearest neighbor comparison is introduced to the survivor selection phase. The algorithm is outlined in Fig. 2.

This approach requires no additional control parameters. Moreover, storage space for the archive together with the computation cost for updating and learning is not necessary. The additional computational time for searching the nearest neighbor in the population is normally negligible compared to the overall computational time taken to solve the optimization problem. This is because it is assumed here that the computational time for function evaluations is so large, and the size of the population so small, that the time taken for the detection of the PUT vector will be comparatively small.

4 Experiments

4.1 Test functions

To demonstrate the performance of the proposed DE-NNC, we employed nine well-known benchmark functions, which were also studied in different researches [15–20]. Details of the functions and their search space, where n is the dimension of the decision vector, are given in Table 1.

4.2 Experimental conditions

In these experiments, the algorithm DE/rand/1/bin (binary crossover and random mutation with one pair of individuals) was adopted as the base algorithm. The parameter settings for optimization are given in Table 2. For tests with the functions f_1, f_2, f_3, f_4 and f_5 parameter values are adopted from [18]. The terminate condition for the optimization process of these functions is when the objective function value $<$10e$-$03, or when the number of fitness evaluations, NFE, exceeds 100,000 (assume that we have a computational budget of 100,000 fitness evaluations). For comparison purpose, the experimental conditions for f_6, f_7, f_8 and f_9 are exactly the same as those in the study by Park and Lee [15]. The dimension of the search space is 10 and 50 for the first five test functions and 2 for the other four functions which have fixed dimension. For each function and each algorithm, 25 random runs were executed.

Fig. 2 DE-NNC algorithm outline

```
Population initialization;
Fitness evaluation;
While termination condition not satisfied
        For k = 1 to Population size do
                Create trial vector;
                Search for its nearest neighbor in the population;
                If the nearest neighbor vector is not worse than the target vector
                Then evaluate fitness of the trial vector;
                        If the trial vector is better than the target vector
                        Then replace the target vector by the trial vector in the next generation;
                        End If
                End If
        End For
End While
```

Table 1 Test functions

Test problem	Function	Decision space
Sphere	$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2;\ f_{\min}(0) = 0$	$x_i \in [-5.12, 5.12]$
Rosenbrock	$f_2(\mathbf{x}) = \sum_{i=2}^{n} 100 \times (x_1 - x_i^2)^2 + (1 - x_i)^2;\ f_{\min}(1) = 0$	$x_i \in [-2.048, 2.048]$
Rastrigin	$f_3(\mathbf{x}) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i));\quad f_{\min}(0) = 0$	$x_i \in [-5.12, 5.12]$
Ackley	$f_4(\mathbf{x}) = 20 + e - 20\exp\left(-0.2\frac{1}{n}\sum_{i=1}^{n} x_i^2\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right);$ $f_{\min}(0) = 0$	$x_i \in [-7, 7]$
Alpine	$f_5(x) = \sum_{i=1}^{n} \lvert x_i \sin(x_i) + 0.1x_i\rvert;\ f_{\min}(0) = 0$	$x_i \in [-10, 10]$
Six-hump Camel-back	$f_6(x_1, x_2) = 4x_1^2 - 2.1x_1^4 + \frac{x_1^6}{3} + x_1x_2 - 4x_2^2 + 4x_2^4;$ $f_{\min} = -1.0316285$	$x_i \in [-5, 5]$
Branin	$f_7(x_1, x_2) = 10 + \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1);$ $f_{\min} = 0.39789$	$x_1 \in [-5, 10],$ $x_2 \in [0, 15]$
Goldstein–price	$f_8(x_1, x_2) = \left[1 + (1 + x_1 + x_2)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right]$ $\times \left[30 + (2x_1 - 3x_2)^2\left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right];$ $f_{\min}(0, -1) = 3$	$x_i \in [-2, 2]$
Bukin	$f_9(x_1, x_2) = 100\sqrt{\lvert x_2 - 0.01x_1^2\rvert} + 0.01\lvert x_1 + 10\rvert;\ f_{\min}(-10, 1) = 0$	$x_1 \in [-15, -5],$ $x_2 \in [-3, 3]$

Table 2 Parameter setting in experiments

Func.	*NP*	*F*	*CR*	Max. generation ($G_{\max}$)	Max. evaluation ($N_{\max}$)	Stop condition
f_1	80	0.6	0.95	–	100,000	$f_{\min} < 10^{-3}$ or NFE $> N_{\max}$
f_2	80	0.6	0.95	–	100,000	$f_{\min} < 10^{-3}$ or NFE $> N_{\max}$
f_3	80	0.6	0.95	–	100,000	$f_{\min} < 10^{-3}$ or NFE $> N_{\max}$
f_4	80	0.6	0.95	–	100,000	$f_{min} < 10^{-3}$ or NFE $> N_{\max}$
f_5	80	0.6	0.95	–	100,000	$f_{min} < 10^{-3}$ or NFE $> N_{\max}$
f_6	50	0.5	0.9	350		$G > G_{\max}$
f_7	50	0.5	0.9	350		$G > G_{\max}$
f_8	50	0.5	0.9	350		$G > G_{\max}$
f_9	50	0.5	0.9	350		$G > G_{\max}$

4.3 Results and discussion

4.3.1 Test functions f_1, f_2, f_3, f_4 and f_5

For these functions, the proposed DE-NCC was tested and compared with the DE using estimated comparison (denoted as DE-EC) and conventional DE. Here, the DE-EC was based on the potential model of approximation with the reference value given by the standard deviation of approximation values [17]. Two values of error margin parameter were considered for DE-EC, which are 0.01 and 0.1.

It was found that with n =10, all tests stopped before the number of fitness evaluations reached $N_{\max}$. Thus, we assessed the performance of the tested algorithms based on two measurement criteria: number of actual function evaluations (NFE) until the fitness value <10e−03 and the number of functions' skip.

Table 3 shows the average results of 25 random runs for each algorithm and each function with n =10. The column "Func." shows the optimized function name and "Method" shows the algorithm, where "NCC" means DE with nearest neighbor comparison, "DE" means original DE/rand/1/bin and others mean DE with estimated comparison using fixed error margin parameter values. The column "eval" and "skip" show the total number of fitness evaluations and the total number of function skips, respectively. The column "success-eval" shows the number of successful evaluations where the child is better than the parent, while the column "success-skip" shows the number of success function skips where the skipped child is worse than the parent. Thus, the success rate of evaluation and success rate of skip are given in the corresponding columns "rate". The column "reduction" shows the percentage of reduction of function evaluations compared with DE.

It was shown that DE-NCC achieved the reduction of function evaluations of 44.51 % for f_1, 37.51 % for f_2, 21.57 % for f_3, 44.31 % for f_4 and 21.17 % for f_5, compared to DE. These results were better than the results by DE-EC using

Table 3 Comparison of the number of fitness evaluations among DE with nearest neighbor comparison, DE with the estimated comparison method and DE

Func.	Method	Function evaluation			Function skip			Reduction (%)
		Eval	Success-eval	Rate (%)	Skip	Success-skip	Rate (%)	
f_1	NNC	7.3596e+03	2.3434e+03	31.84	6.7076e+03	6.3926e+03	95.30	44.51
	EC (0.01)	**5.1019e+03**	2.2033e+03	**43.19**	**8.4245e+03**	7.9809e+03	94.73	**61.54**
	EC (0.1)	1.0731e+04	0.2418e+04	22.53	0.2396e+04	0.2327e+04	**97.12**	19.10
	DE	1.3264e+04	0.2495e+04	18.81	–	–	–	–
f_2	NNC	**1.9595e+04**	0.5124e+04	26.15	1.7650e+04	1.6791e+04	**95.13**	**37.51**
	EC (0.01)	2.1840e+04	0.6830e+04	**31.27**	**4.5005e+04**	3.1479e+04	69.95	30.35
	EC (0.1)	2.7609e+04	0.6166e+04	22.33	1.3447e+04	1.1046e+04	82.14	11.95
	DE	3.1357e+04	0.4771e+04	15.22	–	–	–	–
f_3	NNC	5.8686e+04	0.3536e+04	6.03	**3.4146e+04**	3.3514e+04	**98.15**	21.57
	EC(0.01)	**4.2549e+04**	0.3308e+04	**7.77**	3.1134e+04	3.0297e+04	97.31	**43.13**
	EC (0.1)	6.4574e+04	0.3531e+04	5.47	0.7897e+04	0.7773e+04	**98.43**	13.70
	DE	7.4822e+04	0.3626e+04	4.85	–	–	–	–
f_4	NNC	9.5881e+03	2.8308e+03	29.52	8.8311e+03	8.4147e+03	95.28	44.31
	EC (0.01)	**0.6621e+04**	0.2696e+04	**40.72**	**1.0935e+04**	1.0311e+04	94.29	**61.54**
	EC (0.1)	1.3966e+04	0.2934e+04	21.01	0.3116e+04	0.3028e+04	**97.18**	18.88
	DE	1.7216e+04	0.2988e+04	17.36	–	–	–	–
f_5	NNC	3.6860e+04	0.3435e+04	9.32	**2.5700e+04**	2.4859e+04	**96.73**	21.17
	EC (0.01)	**1.9520e+04**	0.3202e+04	16.40	1.5725e+04	1.4676e+04	93.33	**58.25**
	EC (0.1)	4.0019e+04	0.3358e+04	8.39	0.5018e+04	0.4850e+04	96.65	14.41
	DE	4.6758e+04	0.3431e+04	7.34	–	–	–	–

the margin parameter of 0.1. With the margin parameter of 0.01, DE-EC was superior to DE-NCC in optimization of f_1, f_3, f_4 and f_5. However, for f_2, DE-EC was not better than DE-NCC because it was sometimes trapped by the local minimum. The success rate of skip by DE-NCC was more than 95 % for all five test functions, which was as good as that obtained by DE-EC. It means that the nearest neighbor comparison can efficiently reduce the number of function evaluations and skip only a relatively small number of good trial points in the search.

In the case of $n = 50$, the tests were terminated when the number of fitness evaluations exceeded 100,000. Table 4 shows the optimal results obtained. The columns labeled "average", "best", "worst" and "std" show the average value, the best value, the worst value and the standard deviation of the optimal value in 25 runs, respectively.

For the best average value, DE-NCC found better value than DE, except for f_4. On the other hand, DE-NCC could attain as good results as DE-EC in most cases. Thus, the nearest neighbor comparison method can find better solution and reduce the number of function evaluations effectively.

Figures 3, 4, 5, 6 and 7 show the logarithmic plots of the best function values over the number of function evaluations for functions f_1, f_2, f_3, f_4 and f_5, respectively. Note that the ends of graphs in case n =10 are violated because some runs stopped earlier than other runs when the termination condition was satisfied. In the graphs, thick solid lines and thin solid lines show the optimization process by DE-NNC and DE, respectively. The dashed lines and dotted lines show the optimization process by DE-EC. It can be seen in the figures that the DE-NNC is faster than DE in most cases. Figure 4 clearly shows that for $f_2(n = 10)$ the DE-EC was trapped by the local minimum with the graphs going horizontally. For functions f_4 with $n = 50$, both DE-NNC and DE-EC were trapped by the local minimum as seen in Fig. 6. This explains why the results of DE-NNC and DE-EC were not better than that of DE.

4.3.2 Test function f_6, f_7, f_8 and f_9

These functions were used to examine the performance of DE-NNC and compare with that of DE-EC, DE-kNN and DE-EkNN. For each function, 30 random runs are executed. For each run, the optimization is terminated when the number of generation exceeds 350.

First, the DE-EC was examined with different values of the margin parameter, $\delta = 0.15, 0.10, 0.05, 0.01$. The algorithm DE/rand/1/bin was adopted. The reference value was given by the standard deviation of approximation values. The results are given in Table 5, including the number of func-

Table 4 Comparison of optimal values among DE with nearest neighbor comparison, DE with the estimated comparison method and DE

Func.	Method	Average	Best	Worst	SD
f_1	NNC	0.2387e−08	0.0396e−08	0.9371e−08	0.2126e−08
	EC (0.01)	**0.0257e−09**	**0.0019e−09**	**0.1028e−09**	**0.0266e−09**
	EC (0.1)	0.0322e−03	0.0101e−03	0.1500e−03	0.0275e−03
	DE	0.2208e−03	0.0686e−03	0.6623e−03	0.1505e−03
f_2	NNC	**20.0775**	13.4889	42.1589	7.4079
	EC (0.01)	22.1464	**10.1215**	43.3575	9.2755
	EC (0.1)	23.4219	18.4566	**32.4631**	4.1733
	DE	26.1143	21.5803	33.6443	**3.2898**
f_3	NNC	**313.8919**	**272.4402**	346.6497	17.6752
	EC (0.01)	**313.6343**	280.3830	**332.3530**	13.2411
	EC (0.1)	323.1033	303.3493	339.7008	**10.9903**
	DE	322.7983	276.8880	338.9722	13.4793
f_4	NNC	0.0385	**0.0000**	0.2220	0.0645
	EC (0.01)	0.0830	**0.0000**	0.2955	0.0719
	EC (0.1)	0.0179	**0.0000**	**0.0744**	**0.0324**
	DE	**0.0094**	0.0002	**0.0748**	**0.0246**
f_5	NNC	0.7769	0.0693	12.2751	2.4517
	EC (0.01)	**0.0627**	**0.0556**	**0.0980**	**0.0081**
	EC (0.1)	1.4176	0.1048	6.5734	1.5970
	DE	3.8971	0.2161	12.0804	3.5273

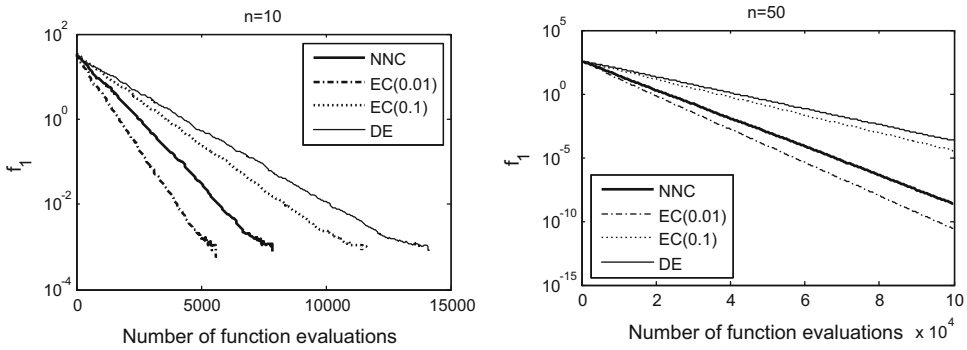


Fig. 3 Optimization of f_1

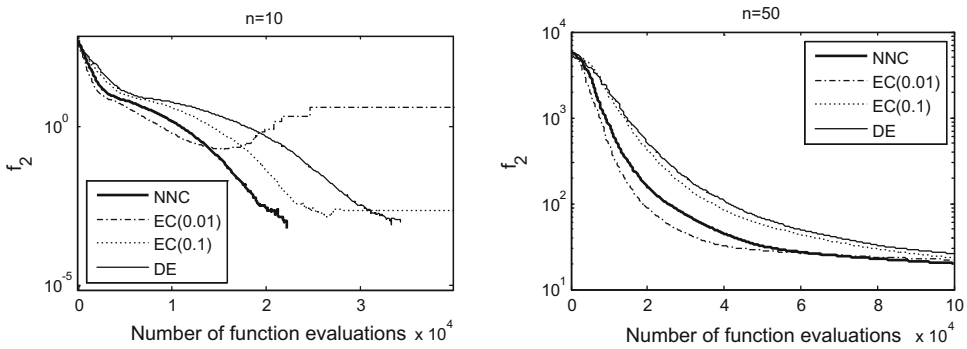


Fig. 4 Optimization of f_2

tion evaluations (NFE) and the function values (FV) optimized. The column "Average" shows the average values of 100 runs. The "Best" and "Worst" columns show the smallest and largest values, respectively. The standard deviations of NFE and FV for 100 runs are given in the column "SD". It is shown that when the value of the margin parameter decreases, the average number of actual function evaluations also decreases. The largest average number of function calls

Fig. 5 Optimization of f_3

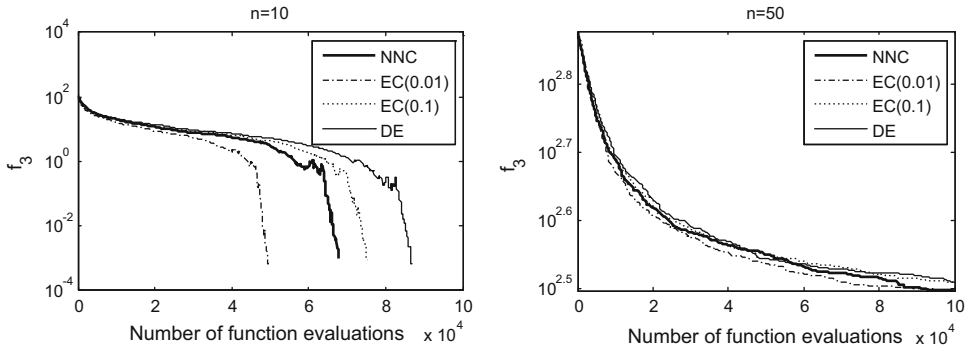


Fig. 6 Optimization of f_4

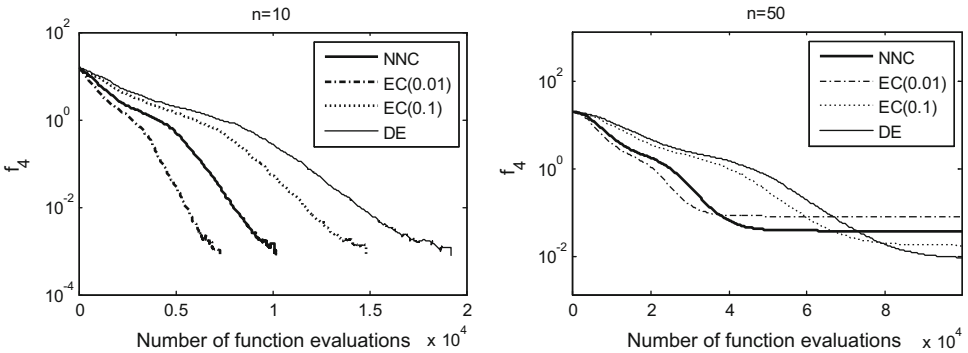
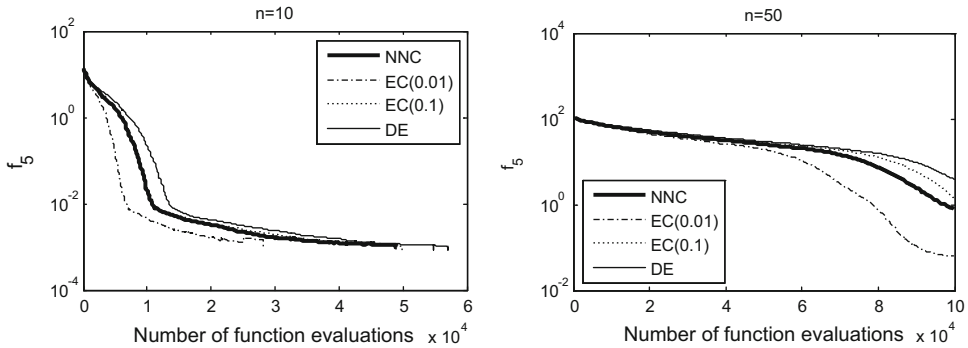


Fig. 7 Optimization of f_5



is 9,631 with $\delta = 0.15$ and the smallest one is 6,233 corresponding to $\delta = 0.01$. However, with smaller value of δ, the optimized function value is worse and its standard deviation is higher, i.e., less accuracy and less stability. On the other hand, higher margin parameters can give better solution and more stability (lower function values and its standard deviations as shown in Table 5). In this study, DE-EC using $\delta = 0.1$ is taken for comparison.

Table 6 shows the results of optimization obtained by DE-NCC and DE-EC, including the optimal function value attained and the number of fitness evaluations after 350 iterations. The results for DE-kNN and DE-EkNN taken from the study by Park and Lee [15] are also listed in Table 6.

Considering the function values optimized, DE-NCC was as good as other algorithms, even better for f_9. Considering the reduction of function evaluations, DE-NNC and DE-EC were not much different. Nevertheless, both methods required less function calls than that by DE-kNN and DE-EkNN. With smaller standard deviation of evaluations for all functions except f_7, DE-NCC was shown to be more stable than DE-EC.

In addition, more computational cost was required by DE-EC, DE-kNN and DE-EkNN, because they do need time for approximation or learning from and updating the archive. This implies that we can get a considerable advantage with DE-NCC.

4.4 Efficiency of DE-NNC

4.4.1 Effect of crossover rate

First, the influence of the crossover rate on the efficiency of the nearest neighbor comparison was examined. Without loss

Table 5 Results of DE using estimated comparison with difference values of margin parameter, δ

δ	Value	Average	Best	Worst	SD
0.15	NFE	9631	3764	11021	1.7113e+03
	FV	**0.0216**	1.3941e−04	0.0476	0.0126
0.1	NFE	8823	2959	10029	1.5817e+03
	FV	0.022	3.3785e−04	0.0494	0.0143
0.05	NFE	7513	3968	8814	1.2849e+03
	FV	0.0244	8.6319e−04	0.0498	0.0156
0.01	NFE	**6233**	3266	7329	875.9635
	FV	0.0285	8.2409e−04	0.0509	0.0160

Table 6 Comparison among DE with nearest neighbor comparison, DE with estimated comparison, DE using k-nearest neighbor predictor and DE using speeded-up k-nearest neighbor estimator

Func.	Method	Optimal value		Fitness evaluation	
		Mean	SD	Mean	SD
f_6	NNC	−1.0316e+00	0.0000e+00	**1.1604e+03**	0.1200e+03
	EC(0.1)	−1.0316e+00	0.0000e+00	**1.1071e+03**	0.1236e+03
	kNN	−1.0316e+00	0.0000e+00	1.0465e+04	0.1613e+03
	EkNN	−1.0316e+00	8.0656e−11	1.0630e+04	0.0000e+00
f_7	NNC	3.9789e−01	0.0000e+00	**1.0143e+03**	0.1910e+03
	EC(0.1)	3.9789e−01	0.0000e+00	**1.0207e+03**	0.1543e+03
	kNN	3.9789e−01	0.0000e+00	1.0475e+04	0.1441e+03
	EkNN	3.9789e−01	2.4938e−08	1.0630e+04	0.0000e+00
f_8	NNC	3.0000e+00	0.0000e+00	**1.9150e+03**	0.0694e+03
	EC(0.1)	3.0000e+00	0.0000e+00	**1.9260e+03**	0.1024e+03
	kNN	3.0000e+00	1.0729e−14	1.0614e+04	0.0237e+03
	EkNN	3.0000e+00	1.7379e−14	1.0630e+04	0.0000e+00
f_9	NNC	**1.8188e−02**	**1.2576e−02**	**8.5055e+03**	0.2010e+03
	EC(0.1)	2.4742e−02	1.8471e−02	0.9326e+04	0.778e+03
	kNN	6.9432e−02	4.8965e−02	1.0630e+04	0.0000e+00
	EkNN	2.3671E−02	2.2373E−02	1.0627e+04	0.0146e+03

of generality, we tested using function f_1 with dimension of decision vector of 10. All parameter settings were the same as in the previous experiment, except that the crossover rate was set with different values, CR = 0.95, 0.5 and 0.1. For each case, 25 runs were executed. Each run stopped when the fitness value <10e−03.

Table 7 shows the optimization results of DE-NCC, DE-EC and DE. The columns labeled "eval", "skip" and "rate" show the total number of evaluations, the number of evaluation skip and the ratio of evaluation skip, respectively. The column "reduce" shows the ratio of the number of times fitness evaluations is reduced compared with DE.

It is seen that when the crossover rate decreases, the reduction ratio is lower and the skip rate also decreases. It means that the nearest neighbor comparison is less efficient with smaller crossover rate. This is also true for DE with estimated comparison as can be seen in Table 7. The explanation for this is that, with small crossover rate, the parent point is close to the trial point and becomes the nearest neighbor of the trial point. Thus, the algorithm judges the trial point as a good trial.

4.4.2 Effect of additional computation time

The DE-NCC can reduce the number of function evaluations in the tested problems compared with standard DE. However, it requires additional time to search for the nearest neighbor point of the trial point in the population. For each trial point, DE-NCC had to compute *NP* distance measure given by Eq. 5. Moreover, in the optimization process, DE-NCC sometimes rejected good trial points (<5 % of the skipped points in our experiment, see Table 3) and took more iterations, thus increasing more time for distance measure. In this study, all test functions were not expensive to evaluate, so

Table 7 Fitness evaluations with different crossover rates for optimization of f_1 ($n = 10$)

CR	Method	Eval	Skip	Rate (%)	Reduction (%)
0.95	NNC	7.3596e+03	6.7076e+03	47.68	44.51
	EC (0.01)	5.1019e+03	8.4245e+03	62.28	61.54
	EC (0.1)	1.0731e+04	0.2396e+04	18.25	19.10
	DE	1.3264e+04	–	–	–
0.5	NNC	7.2574e+03	3.4882e+03	32.46	27.38
	EC (0.01)	4.7351e+03	5.6297e+03	54.32	52.62
	EC (0.1)	8.7105e+03	1.3311e+03	13.26	12.84
	DE	9.9936e+03	–	–	–
0.1	NNC	8.9143e+03	1.0633e+03	10.66	5.73
	EC (0.01)	5.7984e+03	3.8240e+03	39.74	38.68
	EC (0.1)	8.7359e+03	0.8609e+03	8.97	7.62
	DE	9.4560e+03	–	–	–

Table 8 Time consumed and number of generations in the optimization of f_1, f_2, f_3, f_4 and f_5

Func.	Method	G	Time (s)
f_1	DE-NCC	175.84	4.4702
	DE	165.80	0.7027
f_2	DE-NCC	465.56	13.7536
	DE	391.96	1.7510
f_3	DE-NCC	1,160.40	28.1868
	DE	935.28	3.8990
f_4	DE-NCC	230.24	6.0240
	DE	215.20	0.9971
f_5	DE-NCC	782.00	23.1402
	DE	584.48	4.1087

that DE-NCC took more time than DE. Table 8 shows the average time consumed as well as the number of iterations required for optimization of functions f_1, f_2, f_3, f_4 and f_5 with $n = 10$.

It is obvious that this method may be less time consuming and more effective than standard DE only when the cost of function evaluation is much higher than the cost of distance measure.

5 Conclusions

A simple method for reducing the number of function evaluations in differential evolution was proposed. In this method, the function evaluation of a solution is omitted when the fitness of its nearest point in the search population is worse than that of the compared point. The method is named as nearest neighbor comparison (DE-NNC). With the same parameters, the proposed DE-NNC was shown to be able to reduce considerable function evaluations compared with standard DE. It was shown for the test problems that the DE-NNC is competitive to the other DE algorithms using the approximation model, which are DE with estimated comparison, DE-kNN and DE-EkNN. The advantage of DE-NNC is that it requires no additional control parameter as well as external archive and maintains a simple structure as standard DE.

It was shown that the crossover rate has large influence on the performance of DE-NCC. Higher values of crossover rates will be more efficient for function evaluation reduction. Moreover, the nearest neighbor comparison is beneficial only for expensive optimization problems, where the cost of the distance measure between two points is much smaller than the cost of function evaluation.

References

1. Storn, R., Price, K.: Differential Evolution–A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces. International Computer Science Institute, Berkeley (1995)
2. Das, S., Suganthan, P.N.: Differential evolution: a survey of the state-of-the-art. IEEE Trans. Evol. Comput. **15**(1), 4–31 (2011)
3. Jin, Y.: A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput. **9**(1), 3–12 (2005)
4. Khu, S.T., Liu, Y., Savic, D.A.: A fast calibration technique using a hybrid genetic algorithm—neural network approach: application to rainfall-runoff models. In The sixth international conference of hydroinformatics (HIC2004), Singapore (2004)
5. Liu, Y., Khu, S. T.: Automatic calibration of numerical models using fast optimization by fitness approximation. In 2007 International joint conference on neural networks (IJCNN), Orlando, Florida, pp 1073–1078 (2007)
6. Yan, S., Minsker, B.S.: A dynamic meta-model approach to genetic algorithm solution of a risk-based groundwater remediation design model. In American Society of Civil Engineers (ASCE) Envi-

ronmental and Water Resources Institute (EWRI) world water and environmental resources congress 2003 and related symposia, Philadelphia, PA (2003)
7. Giunta, A.A., Watson, L.T., Koehler, J.: A comparison of approximation modeling techniques: polynomial versus interpolating models. In Proceedings of the 7th IAA/USAF/NASA/ISSMO symposium on multidisciplinary analysis and design, pp 392–404 (1998)
8. Simpson, T.W., Mauery, T.M., Korte, J.J., Mistree, F.: Comparison of response surface and kriging models for multidisciplinary design optimization. Am. Inst. Aeronaut. Astronaut. **98**(7), 1–16 (1998)
9. Shyy, W., Tucker, P.K., Vaidyanathan, R.: Response surface and neural network techniques for rocket engine injector optimization. J. Propuls. Power **17**(2), 391–401 (2001)
10. Guimaraes, F.G., Wanner, E.F., Campelo, F., Takahashi, R.H., Igarashi, H., Lowther, D.A., Ramirez, J.A.: Local learning and search in memetic algorithms. In: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, pp. 9841–9848 (2006)
11. Jin, Y., Sendhoff, B.. Reducing fitness evaluations using clustering techniques and neural network ensembles. In Genetic and Evolutionary Computation-GECCO 2004, Springer, Berlin Heidelberg, pp 688–699 (2004)
12. Jin, Y., Olhofer, M., Sendhoff, B. (2000). On evolutionary optimization with approximate fitness functions. In GECCO, pp 786–793
13. Jin, Y., Olhofer, M., Sendhoff, B.: A framework for evolutionary optimization with approximate fitness functions. IEEE Trans Evolution Comput **6**(5), 481–494 (2002)
14. Liu, Y., Sun, F.: A fast differential evolution algorithm using k-Nearest Neighbour predictor. Expert Syst Appl **38**(4), 4254–4258 (2011)
15. Park, S.Y., Lee, J.J.: An efficient differential evolution using speeded-up k-nearest neighbor estimator. Soft Comput. **18**(1), 35–49 (2014)
16. Takahama, T., Sakai, S.: Reducing function evaluations in differential evolution using rough approximation-based comparison. In IEEE congress on evolutionary computation (CEC) 2008, pp 2307–2314 (2008)
17. Takahama, T., Sakai, S.: A comparative study on kernel smoothers in differential evolution with estimated comparison method for reducing function evaluations. In IEEE congress on evolutionary computation (CEC) 2009, pp 1367–1374 (2009)
18. Takahama, T., Sakai, S.: Reducing function evaluations using adaptively controlled differential evolution with rough approximation model. In computational intelligence in expensive optimization problems, pp 111–129, Springer, Berlin Heidelberg (2010)
19. Takahama, T., Sakai, S.: Efficient constrained optimization by the ε constrained differential evolution with rough approximation using kernel regression. In IEEE congress on evolutionary computation (CEC) 2013, pp 1334–1341 (2013)
20. Neri, F., Tirronen, V.: Recent advances in differential evolution: a survey and experimental analysis. Artif. Intel. Rev. **33**(1–2), 61–106 (2010)